3

LUX ET VERITAS

Some Remarks on the Generalised Bareiss and Levinson Algorithms

Ilse Ipsen

Research Report YALEU/DCS/RR-762
February 1990

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

3

Abstract. The Bareiss (or Schur) and Levinson algorithms are the most popular algorithms for solving linear systems with dense $n \times n$ Toeplitz coefficient matrix in $O(n^2)$ arithmetic operations. Both algorithms have been generalised to solve linear systems whose $n \times n$ coefficient matrices A are not necessarily Toeplitz (in $O(n^3)$ operations). We show in this paper that the generalised Levinson algorithm is a direct consequence of the generalised Bareiss algorithm, thereby considerably simplifying its presentation in comparison to previous work.

Some Remarks on the Generalised Bareiss and Levinson Algorithms

Ilse Ipsen

Research Report YALEU/DCS/RR-762
February 1990

The work presented in this paper was supported by DARPA under contract N00014-88-K-0573.

1. Introduction

The solution of linear systems of equations $Ax = b$ with real dense, non-singular coefficient matrix A is usually accomplished by first determining a triangular factorisation of A [9]. There are two types of triangular factorisations: lower-upper and upper-lower. A lower-upper factorisation is given by $A = L\tilde{U}$, where L is a unit lower triangular and $\tilde{U}$ an upper triangular matrix, while an upper-lower factorisation provides $A = U\tilde{L}$, where U is unit upper triangular and $\tilde{L}$ lower triangular (a unit triangular matrix is a triangular matrix with ones on the main diagonal). For non-singular A the factorisations are unique. The number of arithmetic operations required to factor a $n \times n$ matrix is $O(n^3)$.

In Section 2 we give a simple, structural view of these triangular factorisations, which is intended to lead to and facilitate the explanation of the generalised Bareiss algorithm in Section 3. The generalised Bareiss algorithm [5] computes a matrix Q and the two triangular matrices $\tilde{U}$ and $\tilde{L}$ such that

$$Q\begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix}. \qquad (*)$$

As a product of 2×2 elementary transformations, Q is of the form

$$Q = \begin{pmatrix} L_1 & L_2 \\ U_1 & U_2 \end{pmatrix},$$

where $L_1 + L_2$ is unit lower triangular and $U_1 + U_2$ unit upper triangular. The uniqueness of the factorisations implies $L^{-1} = L_1 + L_2$ and $U_1 + U_2 = U^{-1}$. Thus, the generalised Bareiss algorithm computes explicitly the triangular factors $\tilde{U}$ and $\tilde{L}$ of A. The unit triangular factors L^{-1} and U^{-1} of A^{-1} are obtained implicitly: in factored form as the product Q of 2×2 elementary transformations.

In Section 4 the generalised Bareiss algorithm is scaled to yield a symmetric factorisation for symmetric positive-definite matrices A, the so-called Hyperbolic Cholesky algorithm [4].

Upon multiplication of the components in equation (*) by A^{-1} one obtains

$$Q\begin{pmatrix} I \\ I \end{pmatrix} = \begin{pmatrix} L^{-1} \\ U^{-1} \end{pmatrix},$$

where I is the identity matrix. This is the generalised Levinson algorithm [6] of Section 5; it explicitly computes the matrix Q and the unit triangular factors L^{-1} and U^{-1} of A^{-1}. The two triangular factors of A are available implicitly, they may be determined from $\tilde{U} = L^{-1}A$ and $\tilde{L} = U^{-1}A$.

We also discuss how the matrix Q can be used to accomplish forward elimination and back-substitution in order to determine the solution x to the linear system $Ax = b$, when A is symmetric positive-definite.

Section 6 concludes the paper with a combination of the Bareiss and Levinson algorithms for the fast parallel solution of linear systems with persymmetric coefficient matrices. For the special case of Toeplitz matrices, it seems to be identical to an algorithm from [8], but we provide a much simpler description.

In each section, we discuss how the algorithms take advantage of the situation where the coefficient matrix A is a Toeplitz matrix. A $n \times n$ matrix T is a *Toeplitz matrix* if its elements are constant along the diagonals:

$$T = \begin{pmatrix} t_0 & t_1 & \cdots & t_{n-1} \\ t_{-1} & t_0 & \cdots & t_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{-n+1} & t_{-n+2} & \cdots & t_0 \end{pmatrix}$$

Linear systems of equations $Tx = b$ with Toeplitz coefficient matrix T arise, for instance, from time series analysis, Padé approximations, and discretisations of partial differential and integral equations [3]. Because a $n \times n$ Toeplitz matrix contains at most $2n-1$ *different* elements, Toeplitz linear systems can be solved with $O(n^2)$ or fewer arithmetic operations. In this case, the generalised Bareiss and Levinson algorithms reduce to the classical Schur [2, 13] and Levinson [12] algorithms, which perform the solution of $n \times n$ Toeplitz systems in $O(n^2)$ operations.

The purpose of this paper is to provide simple algebraic explanations for the algorithms. We hope that our strategy of first presenting the methods for non-Toeplitz matrices followed by the subsequent restriction to Toeplitz matrices gives a much clearer idea of the underlying structure of the algorithms. The numerical issues associated with the methods, which we ignored in this paper, are discussed in [3, 9]. We only consider here the case of real matrices, but the methods generalise readily to the complex case. Reference [9] contains all the relevant facts about matrix theory.

2. Solution of Linear Systems based on Gaussian Elimination

Let A be a real non-singular coefficient matrix and b the right-hand side vector of the system of simultaneous linear equations $Ax = b$. When A is a dense matrix such systems are usually solved by computing a triangular factorisation of A.

2.1. General Matrices

There are essentially two types of triangular factorisations. If all leading principal submatrices of A are non-singular then the factoriation $A = L\tilde{U}$ exists and is unique, where L is a unit lower triangular matrix (i.e. a triangular matrix with ones on the diagonal) and $\tilde{U}$ is an upper triangular matrix with non-zero diagonal elements. The linear system $Ax = b$ can then be solved in three steps

1. Factorisation $A = L\tilde{U}$
2. Forward Elimination $Lc = b$
3. Backsubstitution $\tilde{U}x = c$.

Alternatively, if all trailing principal submatrices of A are non-singular, then the factorisation $A = U\tilde{L}$ exists and is unique, where U is a unit upper triangular matrix and $\tilde{L}$ is a lower triangular matrix with non-zero diagonal elements, so $Ax = b$ may be solved by computing

$$A = U\tilde{L}, \quad Ud = b, \quad \tilde{L}x = d.$$

Applying Gaussian Elimination (without pivoting) to factor the $n \times n$ matrix A requires $\frac{1}{3}n^3 + O(n^2)$ arithmetic operations, and performing the two subsequent steps by triangular system solution requires another $O(n^2)$ operations.

In many applications the matrix A satisfies a stronger condition than the two above: it is symmetric positive-definite (and so are all its contiguous principal submatrices). In this case, $\tilde{U} = D_{\mathrm{L}}L^T$ and $\tilde{L} = D_{\mathrm{U}}U^T$ and the two symmetric factorisations $A = LD_{\mathrm{L}}L^T$ and $A = UD_{\mathrm{U}}U^T$ exist, where D_{L} and D_{U} are diagonal matrices with positive elements on the main diagonal. Symmetry saves half of the work in the factorisation step, so only $\frac{1}{6}n^3 + O(n^2)$ arithmetic operations are necessary.

2.2. Toeplitz Matrices

The Levinson algorithm exploits the *persymmetry* of Toeplitz matrices T: $T^T = JTJ$, where J is the permutation matrix, also called 'exchange matrix', with ones on the anti-diagonal [9]. Obviously, if T is persymmetric, so is its inverse.

Persymmetry implies that the triangular factors from the two types of factorisations are permutations of each other. In particular, let $\tilde{U} = D_{\mathrm{L}}U_{\mathrm{L}}$, where D_{L} is a diagonal matrix and U_{L} a

$$A = \begin{bmatrix} u & u & u & u \\ \otimes & x & x & x \\ \otimes & x & x & x \\ \otimes & x & x & x \end{bmatrix} \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ \hline & u & u & u \\ & \otimes & x & x \\ & \otimes & x & x \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ \hline & & u & u \\ & & \otimes & x \end{array}\right] \longrightarrow \begin{bmatrix} u & u & u & u \\ & u & u & u \\ & & u & u \\ & & & u \end{bmatrix} = \tilde{U}$$

Figure 1: Evolving Non-Zero Structure of a 4×4 Matrix
During the Computation of $A = L\tilde{U}$.

unit upper triangular matrix, and similarly $\tilde{L} = D_{\mathrm{U}} L_{\mathrm{U}}$, where D_{U} is diagonal and L_{U} unit lower triangular. Then

$$T = L D_{\mathrm{L}} U_{\mathrm{L}} = U D_{\mathrm{U}} L_{\mathrm{U}}, \qquad \text{and} \qquad T = J T^T J = (J L_{\mathrm{U}}^T J)(J D_{\mathrm{U}} J)(J U^T J),$$

implying that $J L_{\mathrm{U}}^T J = L$, $J D_{\mathrm{U}} J = D_{\mathrm{L}}$ and $J U^T J = U_{\mathrm{L}}$.

If T is also symmetric, i.e. $t_i = t_{-i}$ for $1 \leq i \leq n-1$, then

$$JTJ = T = L D_{\mathrm{L}} L^T = U D_{\mathrm{U}} U^T,$$

so $L = JUJ$. In particular, the last row of T is the reverse of its first row, and its last column the reverse of its first column.

The solution methods outlined above are not able to exploit any Toeplitz structure. That is, even though the $n \times n$ matrix T is described by at most $2n-1$ parameters, the operation count for the linear system solution is still $O(n^3)$. In order to explain algorithms that solve Toeplitz systems in $O(n^2)$ arithmetic operations, it is helpful to first consider how Gaussian elimination changes the zero structure of a general matrix during the factorisation process.

2.3. Structural View of the Factorisations

Gaussian elimination (without pivoting) accomplishes the factorisation $A = L\tilde{U}$ by computing successive columns of the unit lower triangular matrix L and successive rows of the upper triangular matrix $\tilde{U}$ by zeroing out successive columns in A. The zero-structure of the matrices occuring in the evolution from the original matrix A to the final upper triangular $\tilde{U}$ are depicted in the 4×4 example of Figure 1. There, x represents an element which is generally non-zero, u an element of the final matrix $\tilde{U}$, a blank a zero element and $\otimes$ an element doomed for elimination in the current step. The partitioning distinguishes those elements in the upper part of the matrix that have ceased to participate in computation. In each step, an appropriate multiple α of the latest row of $\tilde{U}$, which is the one just underneath the horizontal partitioning line, is subtracted from each row beneath it in order to remove the elements in the next column. So, all operations are of the form

$$\text{lower row} = \text{lower row} - \alpha * \text{latest row of } \tilde{U}.$$

In this process no previously introduced zeros are destroyed, because the latest row of $\tilde{U}$ has the same zero structure as all the lower rows to which it is added. Note that in each step all the rows can be modified simultaneously and independently.

The computation of the factorisation $A = U\tilde{L}$ proceeds in a similar manner.

3. The Generalised Bareiss Algorithm

In [2] Bareiss proposed an algorithm for solving square, non-symmetric Toeplitz systems. The process of factoring the matrix in Bareiss algorithm is identical to the Schur algorithm, which is

$$\begin{pmatrix} A \\ A \end{pmatrix} = \left[\begin{array}{cccc} u & u & u & u \\ \otimes & x & x & x \\ y & \otimes & y & y \\ z & z & \otimes & z \\ \hline x & \otimes & x & x \\ y & y & \otimes & y \\ z & z & z & \otimes \\ l & l & l & l \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ \otimes & & x & x \\ y & \otimes & & y \\ \hline x & & \otimes & x \\ y & y & & \otimes \\ l & l & l & \\ l & l & l & l \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ & & u & u \\ \otimes & & & x \\ \hline x & & & \otimes \\ l & l & & \\ l & l & l & \\ l & l & l & l \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ & & u & u \\ & & & u \\ \hline l & & & \\ l & l & & \\ l & l & l & \\ l & l & l & l \end{array}\right] = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix}$$

Figure 2: Evolving Non-Zero Structure of a 4 × 4 Matrix During the Generalised Bareiss Algorithm.

based on [13]; in addition, Bareiss realised that forward elimination could also be accomplished by recursions similar to those used for the factorisation. In [5] it was shown how to generalise this algorithm to non-Toeplitz matrices. In contrast to Gaussian elimination, which computes either $A = L\tilde{U}$ or $A = U\tilde{L}$, the generalised Bareiss algorithm computes factors of both factorisations $A = L\tilde{U}$ and $A = U\tilde{L}$ by working with two copies of the matrix A:

$$Q \begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix} = \left[\begin{array}{ccc} \cdot & \cdots & \cdot \\ & \ddots & \vdots \\ & & \cdot \\ \cdot & & \\ \vdots & \ddots & \\ \cdot & \cdots & \cdot \end{array}\right].$$

The next section explains how to reduce two copies of the original matrix A to an upper triangular matrix and a lower triangular matrix, and the one following argues that these triangular matrices are (in exact arithmetic) identical to the ones obtained from Gaussian elimination.

3.1. Structural View of the Factorisation

In contrast to Gaussian elimination, which eliminates columns, the generalised Bareiss algorithm eliminates diagonals, which turns out to be crucial for the exploitation of Toeplitz properties. This is shown in the 4 × 4 example of Figure 2; there an element of the final matrix $\tilde{L}$ is denoted by l, and the letters x, y, z denote elements that are generally non-zero. The upper copy of A is transformed to upper triangular form and the lower copy to lower triangular form. This is accomplished by 'rotating' in step k rows $i+k$ in the upper matrix with with rows i in the lower matrix (in Figure 2 those rows are made up of identical letters x, y or z) so as to eliminate one element in each of them, namely $(i+k, i)$ and $(i, i+k)$:

$$\begin{pmatrix} \text{row } i+k \text{ in upper matrix} \\ \text{row } i \text{ in lower matrix} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix} \begin{pmatrix} \text{row } i+k \text{ in upper matrix} \\ \text{row } i \text{ in lower matrix} \end{pmatrix}.$$

The two rows are selected to have the same zero structure, so already introduced zeros are preserved. Note that in each step all pairs of rows can be modified simultaneously and independently. The transformation matrix Q is thus made up as a product of these 2 × 2 'rotations'.

We can now express the generalised Bareiss algorithm formally. As a matter of simplicity, we just refer to entire rows of the matrix rather than individual elements and do not show how one can save operations by taking advantage of the increasing number of zero entries in the matrix. Below, the vectors $a_i^{(0)}$ are initialised to be the ith row of the matrix A. From now on, quantities carrying a positive superscript will be associated with the upper matrix, and those with a negative superscript will be associated with the lower matrix. In the kth step, the kth subdiagonal of the upper matrix and the kth superdiagonal of the lower matrix are removed; that is, elements $(i+k, i)$ in the upper matrix and elements $(i, i+k)$ in the lower matrix are removed by appropriately combining the $(i+k)$th row $a_{i+k}^{(k-1)}$ of the upper matrix with the ith row $a_i^{(-k+1)}$ of the lower matrix.

Factorisation in the Generalised Bareiss Algorithm

$$\begin{aligned}
&1 \le i \le n, \qquad a_i^{(0)} = a_i \\
&1 \le k \le n-1,\ 1 \le i \le n-k, \\
&\qquad \alpha_{i,i+k} = a_{i+k,i}^{(k-1)} / a_{i,i}^{(-k+1)}, \quad \beta_{i+k,i} = a_{i,i+k}^{(-k+1)} / a_{i+k,i+k}^{(k-1)} \\
&\qquad \begin{pmatrix} a_{i+k}^{(k)} \\ a_i^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix} \begin{pmatrix} a_{i+k}^{(k-1)} \\ a_i^{(-k+1)} \end{pmatrix}.
\end{aligned}$$

Now the kth rows of $\tilde{U}$ and $\tilde{L}$ are respectively given by $a_k^{(k-1)}$ and $a_k^{(-n+k)}$.

3.2. Correctness of the Factorisation

It remains to explain why indeed the generalised Bareiss algorithm computes triangular factorisations of the matrix A.

As illustrated in the previous section, the transformation matrix Q that reduces the two copies of a A to triangular matrices consists of products of 2×2 'rotations', and 'rotations' belonging to the same step are disjoint. The matrix Q for the 4×4 example in Figure 2 is of the form

$$Q = \left[\begin{array}{cccc|cccc}
1 &&&&&&& \\
& 1 &&&&&& \\
&& 1 &&&&& \\
&&& 1 & x &&& \\ \hline
&&& x & 1 &&& \\
&&&&& 1 && \\
&&&&&& 1 & \\
&&&&&&& 1
\end{array}\right]
\left[\begin{array}{cccc|cccc}
1 &&&&&&& \\
& 1 &&&&&& \\
&& 1 && x &&& \\
&&& 1 && x && \\ \hline
&& x && 1 &&& \\
&&& x && 1 && \\
&&&&&& 1 & \\
&&&&&&& 1
\end{array}\right]
\left[\begin{array}{cccc|cccc}
1 &&&&&&& \\
& 1 && & x &&& \\
&& 1 &&& x && \\
&&& 1 &&& x & \\ \hline
& x &&& 1 &&& \\
&& x &&& 1 && \\
&&& x &&& 1 & \\
&&&&&&& 1
\end{array}\right].$$

The right-most of the three matrices eliminates the first pair of off-diagonals. The x's in the first subdiagonal of its top right part represent the multipliers for removing the first subdiagonal in the upper matrix; similarly the x's in the first superdiagonal of the bottom left part are the multipliers for removing the first superdiagonal in the lower matrix.

Because subdiagonals are removed in the upper matrix and superdiagonals are removed in the lower matrix, the transformation matrices applied at each step consist of two lower triangular matrices in the top half and two upper triangular matrices in the bottom half of the matrix. Due to the particular zero structure of these triangular matrices the product Q of these transformation matrices is shown in [5] to have the form

$$Q = \left[\begin{array}{cccc|cccc}
1 &&&&&&& \\
\vdots & \ddots &&& \cdot &&& \\
\vdots && \ddots && \vdots & \ddots && \\
\cdot & \cdots & \cdots & 1 & \cdot & \cdots & \cdot & \\ \hline
& \cdot & \cdots & \cdot & 1 & \cdots & \cdots & \cdot \\
&& \ddots & \vdots && \ddots && \vdots \\
&&& \cdot &&& \ddots & \vdots \\
&&&&&&& 1
\end{array}\right] = \begin{pmatrix} L_1 & L_2 \\ U_1 & U_2 \end{pmatrix}.$$

If A is a $n \times n$ matrix, then the $2n \times 2n$ matrix Q consists of four $n \times n$ triangular matrices, whereby L_1 is unit lower triangular, L_2 is strictly lower triangular, U_1 is strictly upper triangular, and U_2 is unit upper triangular. But $L_1 + L_2$ is a unit lower triangular matrix, and so is its inverse; similarly $U_1 + U_2$ and its inverse are unit upper triangular matrices. Moreover, in

$$Q\begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} (L_1 + L_2)A \\ (U_1 + U_2)A \end{pmatrix}$$

$(L_1 + L_2)A$ is upper triangular, and $(U_1 + U_2)A$ is lower triangular. Due to the uniqueness of the triangular factorisations we must then have $L = (L_1 + L_2)^{-1}$, $U = (U_1 + U_2)^{-1}$, $(L_1 + L_2)A = \tilde{U}$ and $(U_1 + U_2)A = \tilde{L}$. Consequently,

$$Q\begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} L^{-1}A \\ U^{-1}A \end{pmatrix} = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix}.$$

Thus, the generalised Bareiss determines explicitly the triangular factors $\tilde{U}$ and $\tilde{L}$ of A and implicitly (as the product of the 'rotations' making up Q) the unit triangular factors L^{-1} and U^{-1} of A^{-1}.

In [5] it is proved that the generalised Bareiss algorithm does not break down if all *contiguous* principal submatrices of A are non-singular. This is a more stringent condition than needed for the independent computations of $A = L\tilde{U}$ and $A = U\tilde{L}$ by Gaussian elimination, which only require all *leading* or all *trailing* principal submatrices of A to be non-singular.

3.3. Forward Elimination and Backsubstitution

Instead of solving a triangular system the transformation matrix Q may be used to perform forward elimination, as Bareiss [2] already realised in the Toeplitz case. From the last equation in the previous section we see that the application of Q amounts to a premultiplication by L^{-1} and U^{-1}, thus Q can also be applied to the right-hand side b in order to effect forward elimination

$$Q\begin{pmatrix} b \\ b \end{pmatrix} = \begin{pmatrix} L^{-1}b \\ U^{-1}b \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}.$$

Backsubstitution can now be performed by solving either $\tilde{U}x = c$ or $\tilde{L}x = d$ in $\frac{n^2}{2} + O(n)$ arithmetic operations. Employing a trick from Bareiss [2], one could alternatively determine the upper half of x from $\tilde{U}x = c$ and the lower half from $\tilde{L}x = d$, which would require only $\frac{n^2}{4} + O(n)$ operations.

If A is symmetric positive-definite, the application of Q^T can replace the triangular system solution for backsubstitution. To this end, let I denote the identity matrix of the same size as A, and

$$\begin{aligned} y \equiv (\, I \quad I\,)Q^T \begin{pmatrix} D_{\mathrm{L}}^{-1} & \\ & D_{\mathrm{U}}^{-1} \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} &= (L_1 + L_2)^T D_{\mathrm{L}}^{-1} c + (U_1 + U_2)^T D_{\mathrm{U}}^{-1} d \\ &= L^{-T} D_{\mathrm{L}}^{-1} c + U^{-T} D_{\mathrm{U}}^{-1} d \end{aligned}$$

because $c = L^{-1}b$ and $d = U^{-1}b$, so

$$y = L^{-T} D_{\mathrm{L}}^{-1} L^{-1} b + U^{-T} D_{\mathrm{U}}^{-1} U^{-1} b = A^{-1}b + A^{-1}b = 2A^{-1}b = 2x.$$

In order to obtain x, rather than $2x$, observe that c and d each give rise to one x, and applying the transformations instead to $(\,\lambda c \quad (1-\lambda)d\,)^T$, where λ is any real number, yields

$$y = (\, I \quad I\,)Q^T \begin{pmatrix} D_{\mathrm{L}}^{-1} & \\ & D_{\mathrm{U}}^{-1} \end{pmatrix} \begin{pmatrix} \lambda c \\ (1-\lambda)d \end{pmatrix} = \lambda A^{-1}b + (1-\lambda)A^{-1}b = \lambda x + (1-\lambda)x = x.$$

$$\left[\begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \\ t_{-1} & t_0 & t_1 & t_2 \\ t_{-2} & t_{-1} & t_0 & t_1 \\ t_{-3} & t_{-2} & t_{-1} & t_0 \\ \hline s_0 & s_1 & s_2 & s_3 \\ s_{-1} & s_0 & s_1 & s_2 \\ s_{-2} & s_{-1} & s_0 & s_1 \\ s_{-3} & s_{-2} & s_{-1} & s_0 \end{array}\right] \longrightarrow \left[\begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \\ & t'_0 & t'_1 & t'_2 \\ t'_{-2} & & t'_0 & t'_1 \\ t'_{-3} & t'_{-2} & & t'_0 \\ \hline s'_0 & & s'_2 & s'_3 \\ s'_{-1} & s'_0 & & s'_2 \\ s'_{-2} & s'_{-1} & s'_0 & \\ s_{-3} & s_{-2} & s_{-1} & s_0 \end{array}\right] \longrightarrow \left[\begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \\ & t'_0 & t'_1 & t'_2 \\ & & t''_0 & t''_1 \\ t''_{-3} & & & t''_0 \\ \hline s''_0 & & & s''_3 \\ s''_{-1} & s''_0 & & \\ s'_{-2} & s'_{-1} & s'_0 & \\ s_{-3} & s_{-2} & s_{-1} & s_0 \end{array}\right] \longrightarrow \left[\begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \\ & t'_0 & t'_1 & t'_2 \\ & & t''_0 & t''_1 \\ & & & t'''_0 \\ \hline s'''_0 & & & \\ s''_{-1} & s''_0 & & \\ s'_{-2} & s'_{-1} & s'_0 & \\ s_{-3} & s_{-2} & s_{-1} & s_0 \end{array}\right]$$

Figure 3: Evolving Structure of a 4×4 Toeplitz Matrix
During Bareiss Algorithm; Initially $s_i = t_i$.

In particular, λ may be set to either zero or one so as to require only one of the right-hand sides c or d as input.

For symmetric positive-definite matrices A the generalised Bareiss algorithm permits the solution of the linear system $Ax = b$ with the following three steps:

1. Determine Q such that $Q\begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} \bar{U} \\ \bar{L} \end{pmatrix}$
2. Determine $\begin{pmatrix} c \\ d \end{pmatrix} = Q\begin{pmatrix} b \\ b \end{pmatrix}$
3. Determine $x = (\,I \quad I\,)Q^T\begin{pmatrix} D_L^{-1} & \\ & D_U^{-1} \end{pmatrix}\begin{pmatrix} \lambda c \\ (1-\lambda)d \end{pmatrix}$.

Because the generalised Bareiss algorithm determines a non-symmetric factorisation $\alpha_{i,i+k} \neq \beta_{i+k,i}$, and it can *generally* not take advantage of the symmetry of a matrix in order to reduce the number of arithmetic operations – except if the matrix is a symmetric Toeplitz matrix: then the operation count decreases by fifty percent compared to the non-symmetric Toeplitz case, as shown in the next section.

3.4. Specialisation to Toeplitz Matrices

If the matrix T to be factored is a Toeplitz matrix, then the generalised Bareiss algorithm is identical to the original Bareiss algorithm [2], which exploits the structure of a Toeplitz matrix.

Figure 3 displays the evolution of the structure of the 4×4 Toeplitz matrix

$$T = \begin{pmatrix} t_0 & t_1 & t_2 & t_3 \\ t_{-1} & t_0 & t_1 & t_2 \\ t_{-2} & t_{-1} & t_0 & t_1 \\ t_{-3} & t_{-2} & t_{-1} & t_0 \end{pmatrix}$$

during the course of Bareiss algorithm. Although the triangular factors of a Toeplitz matrix are generally not Toeplitz, the 'working part' of the matrix in Bareiss algorithm is Toeplitz as shown in Figure 3 and below, because identical elements along each diagonal result in identical $\alpha_{i+k,i}$ and $\beta_{i,i+k}$ for each step; this makes it possible to exploit the Toeplitz structure.

In particular, denote the elements of T by $t_{ij} \equiv t_{j-i}$, $1 \leq i,j \leq n$. Rows a_1 and a_n constitute the respective first row of $\bar{U}$ and last row of $\bar{L}$. Since T is Toeplitz, at the beginning of the first step all the elements necessary for further computation are contained in rows a_2 and a_n of the upper matrix and rows a_1 and a_{n-1} of the lower matrix. The multipliers are

$$\alpha_{i,i+1} = \frac{t_{i+1,i}}{t_{i,i}} = \frac{t_1}{t_0} = \alpha_1, \qquad \beta_{i+1,i} = \frac{t_{i,i+1}}{t_{i+1,i+1}} = \frac{t_{-1}}{t_0} = \beta_1, \qquad 1 \leq i \leq n-1.$$

These multipliers are used in the rotations, as above, to construct rows $a_{i+1}^{(1)}$ of the upper matrix and rows $a_i^{(-1)}$ of the lower matrix, $1 \le i \le n-1$. Hence rows $3, \dots, n$ of the upper matrix and rows $1, \dots, n-2$ of the lower matrix retain their Toeplitz structure, and rows $a_3^{(1)}$, $a_n^{(1)}$, $a_{n-2}^{(-1)}$ and $a_1^{(-1)}$ encompass all necessary information for further computation, see Figure 3.

In general at the beginning of step k, rows $k+1, \dots, n$ of the upper matrix as well as rows $1, \dots, n-k$ of the lower matrix have Toeplitz structure. If we represent the upper and lower triangular parts of the two matrices by separate vectors

$$b_{k+1}^{(k-1)} = \begin{pmatrix} 0 & \dots & 0 & a_{k+1,k+1}^{(k-1)} & \dots & a_{k+1,n}^{(k-1)} \end{pmatrix}, \quad b_n^{(k-1)} = \begin{pmatrix} a_{n,1}^{(k-1)} & \dots & a_{n,n-k}^{(k-1)} & 0 & \dots & 0 \end{pmatrix}$$
$$b_1^{(-k+1)} = \begin{pmatrix} 0 & \dots & 0 & a_{1,k+1}^{(-k+1)} & \dots & a_{1,n}^{(-k+1)} \end{pmatrix}, \quad b_{n-k}^{(-k+1)} = \begin{pmatrix} a_{n-k,1}^{(-k+1)} & \dots & a_{n-k,n-k}^{(-k+1)} & 0 & \dots & 0 \end{pmatrix},$$

then $b_{k+1}^{(k-1)}$ and $b_n^{(k-1)}$ contain all the distinct elements in the respective upper and lower triangular parts of the upper matrix, while $b_{n-k}^{(-k+1)}$ and $b_1^{(-k+1)}$ contain all distinct elements in the respective lower and upper triangular parts of the lower matrix, see Figure 3. Thus, the following two 'rotations' embody all the distinct computations on the Toeplitz matrix during step k:

$$\begin{pmatrix} b_{k+1}^{(k)} \\ b_1^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} b_{k+1}^{(k-1)} \\ b_1^{(-k+1)} \end{pmatrix}, \qquad \begin{pmatrix} b_n^{(k)} \\ b_{n-k}^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} b_n^{(k-1)} \\ b_{n-k}^{(-k+1)} \end{pmatrix}.$$

The transition to the next step is accomplished by considering the next lower row in the upper matrix and the next higher row in the lower matrix:

$$b_{k+2}^{(k)} = b_{k+1}^{(k)} Z, \quad b_{n-k+1}^{(k)} = b_{n-k}^{(k)} Z^T, \qquad \text{where } Z = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ & & & 0 \end{pmatrix}.$$

This leads to the following form of the factorisation process in Bareiss Algorithm, which is also called 'Schur algorithm' [13]. Below, 0_k denotes a row vector consisting of k zeros, and the rows $b_i^{(j)}$ are represented by their individual elements.

Schur Algorithm: Factorisation Part of Bareiss Algorithm

$$\begin{pmatrix} t_0^{(0)} & t_1^{(0)} & \dots & t_{n-1}^{(0)} \end{pmatrix} = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \end{pmatrix}$$
$$\begin{pmatrix} t_{-n+1}^{(0)} & \dots & t_{-1}^{(0)} & t_0^{(0)} \end{pmatrix} = \begin{pmatrix} t_{-n+1} & \dots & t_{-1} & t_0 \end{pmatrix}$$
$$1 \le k \le n-1, \qquad \alpha_k = t_{-k}^{(k-1)} / t_0^{(-k+1)}, \quad \beta_k = t_k^{(-k+1)} / t_0^{(k-1)}$$
$$\begin{pmatrix} 0_k & t_0^{(k)} & t_1^{(k)} & \dots & t_{n-k-1}^{(k)} \\ 0_k & 0 & t_{k+1}^{(-k)} & \dots & t_{n-1}^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} 0_k & t_0^{(k-1)} & t_1^{(k-1)} & \dots & t_{n-k-1}^{(k-1)} \\ 0_k & t_k^{(-k+1)} & t_{k+1}^{(-k+1)} & \dots & t_{n-1}^{(-k+1)} \end{pmatrix}.$$
$$\begin{pmatrix} t_{-n+1}^{(k)} & \dots & t_{-k-1}^{(k)} & 0 & 0_k \\ t_{-n+k+1}^{(-k)} & \dots & t_{-1}^{(-k)} & t_0^{(-k)} & 0_k \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} t_{-n+1}^{(k-1)} & \dots & t_{-k-1}^{(k-1)} & t_{-k}^{(k-1)} & 0_k \\ t_{-n+k+1}^{(-k+1)} & \dots & t_{-1}^{(-k+1)} & t_0^{(-k+1)} & 0_k \end{pmatrix}.$$

Now the kth rows of $\tilde{U}$ and $\tilde{L}$ are respectively given by

$$\begin{pmatrix} 0_k & t_0^{(k)} & t_1^{(k)} & \dots & t_{n-k-1}^{(k)} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} t_{-n+k+1}^{(-k)} & \dots & t_{-1}^{(-k)} & t_0^{(-k)} & 0_k \end{pmatrix}.$$

The recursions for forward elimination and backsubstitution are similar to those in the factorisation; see also [4].

If the Toeplitz matrix T is symmetric positive-definite, then $\alpha_1 = \beta_1$ and, as explained in Section 2, the last row of the Toeplitz matrix is the reverse of the first one, so its computation may be omitted, hence reducing the operation count by fifty percent. The Schur algorithm for this case follows:

Schur Algorithm for Symmetric Positive-Definite Matrices

$$\begin{aligned}
&\begin{pmatrix} t_0^{(0)} & t_1^{(0)} & \dots & t_{n-1}^{(0)} \end{pmatrix} = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \end{pmatrix} \\
&1 \le k \le n-1, \qquad \rho_k = t_k^{(-k+1)}/t_0^{(k-1)}, \\
&\quad \begin{pmatrix} t_0^{(k)} & t_1^{(k)} & \dots & t_{n-k-1}^{(k)} \\ 0 & t_{k+1}^{(-k)} & \dots & t_{n-1}^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\rho_k \\ -\rho_k & 1 \end{pmatrix} \begin{pmatrix} t_0^{(k-1)} & t_1^{(k-1)} & \dots & t_{n-k-1}^{(k-1)} \\ t_k^{(-k+1)} & t_{k+1}^{(-k+1)} & \dots & t_{n-1}^{(-k+1)} \end{pmatrix}
\end{aligned}$$

4. The Hyperbolic Cholesky Algorithm

The hyperbolic Cholesky algorithm [4] (see also [1, 7]) computes both Cholesky factorisations $A = \mathcal{U}\mathcal{U}^T$ and $A = \mathcal{L}\mathcal{L}^T$, where $\mathcal{U}^T = D_{\mathrm{U}}^{1/2} U^T$ and $\mathcal{L}^T = D_{\mathrm{L}}^{1/2} L^T$, of the symmetric positive-definite matrix A. If D is a diagonal matrix whose diagonal equals the diagonal of A, then

$$Q_{\mathrm{H}} \begin{pmatrix} D^{-1/2} & \\ & D^{-1/2} \end{pmatrix} \begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} \mathcal{L}^T \\ \mathcal{U}^T \end{pmatrix} = \begin{pmatrix} D_{\mathrm{L}}^{1/2} L^T \\ D_{\mathrm{U}}^{1/2} U^T \end{pmatrix}.$$

4.1. Connection to the Generalised Bareiss Algorithm

Premultiplying both sides of the equation by $\begin{pmatrix} D_{\mathrm{L}}^{1/2} & \\ & D_{\mathrm{U}}^{1/2} \end{pmatrix}$ gives, due to the uniqueness of the factorisation, the following relations between the quantities computed by the Bareiss and Hyperbolic Cholesky algorithms:

$$\begin{pmatrix} D_{\mathrm{L}}^{1/2} & \\ & D_{\mathrm{U}}^{1/2} \end{pmatrix} Q_{\mathrm{H}} \begin{pmatrix} D^{-1/2} & \\ & D^{-1/2} \end{pmatrix} \begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} D_{\mathrm{L}} L^T \\ D_{\mathrm{U}} U^T \end{pmatrix} = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix}$$

and

$$\begin{pmatrix} D_{\mathrm{L}}^{1/2} & \\ & D_{\mathrm{U}}^{1/2} \end{pmatrix} Q_{\mathrm{H}} \begin{pmatrix} D^{-1/2} & \\ & D^{-1/2} \end{pmatrix} = Q.$$

Thus, as also shown in [5], the Hyperbolic Cholesky algorithm is a 'scaled' version of the generalised Bareiss algorithm.

Instead of the 'rotations' of the generalised Bareiss algorithm, the Hyperbolic Cholesky algorithm uses hyperbolic rotations to eliminate a pair of elements. In particular, if the generalised Bareiss algorithm removes element $(i+k, i)$ in the upper matrix and element $(i, i+k)$ in the lower matrix by applying

$$\begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix}$$

to rows $i+k$ and i, then the Hyperbolic Cholesky algorithm removes elements in the same positions via the hyperbolic rotation

$$\frac{1}{\sqrt{1-\rho_{i+k,i}^2}}\begin{pmatrix} 1 & -\rho_{i+k,i} \\ -\rho_{i+k,i} & 1 \end{pmatrix}$$

where $\rho_{i+k,i} = \sqrt{\alpha_{i,i+k}\beta_{i+k,i}}$. This can easily be seen in a 2×2 example, where

$$A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

The generalised Bareiss algorithm computes

$$\left(\begin{array}{cc|cc} 1 & & & \\ & 1 & -\alpha & \\ \hline & -\beta & 1 & \\ & & & 1 \end{array}\right)\left(\begin{array}{cc} a & b \\ b & c \\ \hline a & b \\ b & c \end{array}\right) = \left(\begin{array}{cc} a & b \\ 0 & c-b^2/a \\ \hline a-b^2/c & 0 \\ b & c \end{array}\right),$$

where $\alpha = b/a$ and $\beta = b/c$. The Hyperbolic Cholesky algorithm computes

$$\frac{1}{\sqrt{1-\rho^2}}\left(\begin{array}{cc|cc} 1 & & & \\ & 1 & -\rho & \\ \hline & -\rho & 1 & \\ & & & 1 \end{array}\right)\left(\begin{array}{cc} \sqrt{a} & b/\sqrt{a} \\ b/\sqrt{c} & \sqrt{c} \\ \hline \sqrt{a} & b/\sqrt{a} \\ b/\sqrt{c} & \sqrt{c} \end{array}\right) = \left(\begin{array}{cc} \sqrt{a} & b/\sqrt{a} \\ 0 & \sqrt{c-b^2/a} \\ \hline \sqrt{a-b^2/c} & 0 \\ b/\sqrt{c} & \sqrt{c} \end{array}\right),$$

where $\rho = b/\sqrt{ac}$. Thus, $\rho = \sqrt{\alpha\beta}$.

The process of forward elimination and backsubstitution proceeds as in the generalised Bareiss but with Q replaced by

$$Q_{\mathrm{H}}\begin{pmatrix} D^{-1/2} & \\ & D^{-1/2} \end{pmatrix}.$$

As a consequence, the Hyperbolic Cholesky algorithm gives rise to a scaled Schur algorithm based on rotations

$$\frac{1}{\sqrt{1-\rho_k^2}}\begin{pmatrix} 1 & -\rho_k \\ -\rho_k & 1 \end{pmatrix}.$$

4.2. View of Hyperbolic Cholesky Factorisation as a Downdating Process

Unlike the generalised Bareiss algorithm the Hyperbolic Cholesky algorithm can take advantage of the symmetry of non-Toeplitz matrices. If only one Cholesky factor is desired, it suffices to compute

$$Q_{\mathrm{H}}\begin{pmatrix} D^{-1/2} & \\ & D^{-1/2} \end{pmatrix}\begin{pmatrix} A_+ \\ A_\# \end{pmatrix} = \begin{pmatrix} \mathcal{L}^T \\ 0 \end{pmatrix},$$

where A_+ consists of the upper triangle of A including the diagonal, and $A_\#$ consists of the strict upper triangle of A excluding the diagonal. As a product of hyperbolic rotations, the matrix Q_{H} is pseudo-orthogonal, that is,

$$Q_{\mathrm{H}}^T\begin{pmatrix} I & \\ & -I \end{pmatrix}Q_{\mathrm{H}} = \begin{pmatrix} I & \\ & -I \end{pmatrix}.$$

$$\begin{pmatrix} V \\ W \end{pmatrix} = \left[\begin{array}{cccc} u & u & u & u \\ & x & x & x \\ & & y & y \\ & & & z \\ \hline & \otimes & x & x \\ & & \otimes & y \\ & & & \otimes \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ & & x & x \\ & & & y \\ \hline & & \otimes & x \\ & & & \otimes \\ & & & \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ & & u & u \\ & & & x \\ \hline & & & \otimes \\ & & & \\ & & & \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & u & u & u \\ & & u & u \\ & & & u \\ \hline & & & \\ & & & \\ & & & \\ & & & \end{array}\right]$$

(a) Removing Successive Diagonals.

$$\begin{pmatrix} V \\ W \end{pmatrix} = \left[\begin{array}{cccc} u & u & u & u \\ & x & x & x \\ & & x & x \\ & & & y \\ \hline & x & x & x \\ & & x & x \\ & & & \otimes \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & x & x & x \\ & & y & y \\ & & & x \\ \hline & x & x & x \\ & & \otimes & y \\ & & & \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & x & x & x \\ & & x & x \\ & & & y \\ \hline & x & x & x \\ & & & \otimes \\ & & & \\ & & & \end{array}\right] \longrightarrow \left[\begin{array}{cccc} u & u & u & u \\ & y & y & y \\ & & x & x \\ & & & x \\ \hline & \otimes & y & y \\ & & & \\ & & & \\ & & & \end{array}\right] \longrightarrow \cdots$$

(b) Removing Successive Rows.

Figure 4: Two Different Orders of Rotations in the Hyperbolic Cholesky Algorithm Applied to a 4 × 4 Matrix.

Multiplying the previous equation by its transpose and exploiting the pseudo-orthogonality of Q_{H} yields $\mathcal{L}\mathcal{L}^T = VV^T - WW^T$, where $V = D^{-1/2}A_+$ is an upper triangular matrix with positive diagonal elements, and $W = D^{-1/2}A_\#$ is a strictly upper triangular matrix. Denote by l_i the columns of W, so $\mathcal{L}\mathcal{L}^T = VV^T - \sum_i l_i l_i^T$; and $V^{(0)} \equiv V$. We can view the Hyperbolic Cholesky algorithm as computing a sequence of Cholesky downdating problems: each problem consists of premultiplying $\begin{pmatrix} V^{(i)} \\ l_i^T \end{pmatrix}$ by hyperbolic rotations resulting in $\begin{pmatrix} V^{(i+1)} \\ 0 \end{pmatrix}$ and

$$V^{(i+1)}V^{(i+1)^T} = V^{(i)}V^{(i)^T} - l_i l_i^T$$

such that $V^{(i+1)}$ is an upper triangular matrix with positive diagonal elements, see for example [11]. The Hyperbolic Cholesky algorithm computes the rotations in the same order as the generalised Bareiss algorithm. This is also the order that a downdating process would choose, as Figure 4 illustrates. Instead of removing successive diagonals, as shown in Figure 4(a), one can also pipeline the rotations so as to remove successive rows; see Figure 4(b).

Reducing the Hyperbolic Cholesky algorithm to a sequence of downdating problems may facilitate its round-off error analysis due to the availability of round-off error analyses for downdating problems.

5. The Generalised Levinson Algorithm

The first explicit algorithm for solving $n \times n$ Toeplitz systems with $O(n^2)$ operations was introduced by Levinson in 1947 [12]. More recently, Levinson's algorithm was extended to non-Toeplitz matrices by Delsarte, Genin and Kamp [6], just as Bareiss algorithm [2] was extended to the generalised Bareiss algorithm [5]. In previous work [10] we have shown that the Schur algorithm can be derived as the result of trying to increase the degree of parallelism in the Levinson algorithm (i.e. by getting rid of the inner products). Conversely, we will now show that the generalised Levinson algorithm is a simple consequence of the generalised Bareiss algorithm. Our version of the generalised Levinson algorithm is simpler and more intuitive than the one in [6] as it does without the exchange matrix J, and it establishes a direct connection to the other factorisation methods.

5.1. From the Generalised Bareiss to the Generalised Levinson Algorithm

The generalised Bareiss algorithm computes

$$Q\begin{pmatrix} A \\ A \end{pmatrix} = \begin{pmatrix} \tilde{U} \\ \tilde{L} \end{pmatrix},$$

where $A = L\tilde{U}$ and $A = U\tilde{L}$ are the triangular factorisations of a non-singular matrix A. Postmultiplying both sides of the equation by $\begin{pmatrix} A^{-1} & \\ & A^{-1} \end{pmatrix}$ gives

$$Q\begin{pmatrix} I \\ I \end{pmatrix} = \begin{pmatrix} \tilde{U}A^{-1} \\ \tilde{L}A^{-1} \end{pmatrix} = \begin{pmatrix} L^{-1} \\ U^{-1} \end{pmatrix}$$

since $A^{-1} = \tilde{U}^{-1}L^{-1}$ and $A^{-1} = \tilde{L}^{-1}U^{-1}$. Thus, by applying the matrix Q to the identity matrix the generalised Levinson algorithm determines explicitly the unit triangular factors L^{-1} and U^{-1} of A^{-1} and implicitly (by forming $\tilde{U} = L^{-1}A$ and $\tilde{L} = U^{-1}A$) the non-unit triangular factors of A.

Now we consider how the elements of Q are actually determined by the generalised Levinson algorithm. If initially $a_i^{(0)} = a_i$, $1 \le i \le n$, where a_i is the ith row of A then the kth step of the generalised Bareiss algorithm combines the $(i+k)$th row $a_{i+k}^{(k-1)}$ of the upper matrix with the ith row $a_i^{(-k+1)}$ of the lower matrix:

$$\begin{pmatrix} a_{i+k}^{(k)} \\ a_i^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix} \begin{pmatrix} a_{i+k}^{(k-1)} \\ a_i^{(-k+1)} \end{pmatrix}, \qquad 1 \le i \le n-k,$$

where

$$\alpha_{i,i+k} = a_{i+k,i}^{(k-1)}/a_{i,i}^{(-k+1)}, \qquad \beta_{i+k,i} = a_{i,i+k}^{(-k+1)}/a_{i+k,i+k}^{(k-1)}$$

to remove element $(i+k, i)$ in the upper matrix and element $(i, i+k)$ in the lower matrix. Eventually, $a_k^{(k-1)}$ is the kth row of $\tilde{U}$ and $a_k^{(-n+k)}$ is the kth row of $\tilde{L}$.

This implies that the kth row of $\tilde{U}A^{-1} = L^{-1}$ is $\psi_k^{(k-1)} \equiv a_k^{(k-1)}A^{-1}$ and the kth row of $\tilde{L}A^{-1} = U^{-1}$ is $\psi_k^{(-n+k)} \equiv a_k^{(-n+k)}A^{-1}$. In particular, $\psi_1^{(0)} = a_1A^{-1} = e_1^T$ and $\psi_n^{(0)} = a_nA^{-1} = e_n^T$, where e_i are the $n \times 1$ canonical vectors with a one in position i and zeros everywhere else. This motivates

$$\psi_i^{(0)} \equiv a_iA^{-1} = e_i^T, \qquad 1 \le i \le n.$$

If

$$\psi_{i+k}^{(k)} = a_{i+k}^{(k)}A^{-1}, \qquad \psi_i^{(-k)} = a_i^{(-k)}A^{-1}, \qquad 1 \le i \le n-k,$$

then it follows by induction that

$$\alpha_{i,i+k} = \psi_{i+k}^{(k-1)}Ae_i/\psi_i^{(-k+1)}Ae_i, \qquad \beta_{i+k,i} = \psi_i^{(-k+1)}Ae_{i+k}/\psi_{i+k}^{(k-1)}Ae_{i+k},$$

and

$$\begin{pmatrix} \psi_{i+k}^{(k)} \\ \psi_i^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix} \begin{pmatrix} \psi_{i+k}^{(k-1)} \\ \psi_i^{(-k+1)} \end{pmatrix}, \qquad 1 \le i \le n-k.$$

Note that only elements $i \ldots i+k$ of $\psi_{i+k}^{(k)}$ and $\psi_i^{(-k)}$ may be non-zero. To keep the presentation simple, we do not take advantage of this fact in the description of the generalised Levinson algorithm below.

The above derivation shows that, in contrast to the presentation of the generalised Levinson algorithm in [6], there is no need for the exchange matrix J. Its occurrence in the symmetric Levinson algorithm is an artefact of the symmetric Toeplitz structure, as explained in the next section.

We further observe that the inner products in the denominators of $\alpha_{i,i+k}$ and $\beta_{i+k,i}$ can be replaced by recursive equations, thus saving about $O(2k)$ operations in step k of the generalised Levinson algorithm. Define the denominators of $\alpha_{i,i+k}$ and $\beta_{i+k,i}$ by

$$d_i^{(-k+1)} \equiv \psi_i^{(-k+1)} A e_i, \quad d_{i+k}^{(k-1)} \equiv \psi_{i+k}^{(k-1)} A e_{i+k},$$

so that

$$\alpha_{i,i+k} = \psi_{i+k}^{(k-1)} A e_i / d_i^{(-k+1)}, \quad \beta_{i+k,i} = \psi_i^{(-k+1)} A e_{i+k} / d_{i+k}^{(k-1)}.$$

We can now derive recursive equations for $d_i^{(-k)}$ and $d_{i+k+1}^{(k)}$ in terms of $d_i^{(-k+1)}$ and $d_{i+k+1}^{(k-1)}$: initially

$$d_i^{(0)} = \psi_i^{(0)} A e_i = e_i^T A e_i = a_{ii}, \qquad 1 \leq i \leq n.$$

From the above definition and the computation of $\psi_i^{(-k)}$ it follows that

$$\begin{aligned} d_i^{(-k)} &= \psi_i^{(-k)} A e_i = \psi_i^{(-k+1)} A e_i - \beta_{i+k,i} \psi_{i+k}^{(k-1)} A e_i = d_i^{(-k+1)} - \alpha_{i,i+k}\beta_{i+k,i} d_i^{(-k+1)} \\ &= (1 - \alpha_{i,i+k}\beta_{i+k,i}) d_i^{(-k+1)}. \end{aligned}$$

Similary, one shows that

$$d_{i+k}^{(k)} = (1 - \alpha_{i,i+k}\beta_{i+k,i}) d_{i+k}^{(k-1)}.$$

Our formulation of the generalised Levinson algorithm is summarised below.

Factorisation in the Generalised Levinson Algorithm

$$\begin{aligned} &1 \leq i \leq n, \qquad \psi_i^{(0)} = e_i^T, \quad d_i^{(0)} = a_{ii} \\ &1 \leq k \leq n-1, \ 1 \leq i \leq n-k, \\ &\quad \alpha_{i,i+k} = \psi_{i+k}^{(k-1)} A e_i / d_i^{(-k+1)}, \qquad \beta_{i+k,i} = \psi_i^{(-k+1)} A e_{i+k} / d_{i+k}^{(k-1)} \\ &\quad d_{i+k}^{(k)} = (1 - \alpha_{i,i+k}\beta_{i+k,i}) d_{i+k}^{(k-1)}, \qquad d_i^{(-k)} = (1 - \alpha_{i,i+k}\beta_{i+k,i}) d_i^{(-k+1)} \\ &\quad \begin{pmatrix} \psi_{i+k}^{(k)} \\ \psi_i^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_{i,i+k} \\ -\beta_{i+k,i} & 1 \end{pmatrix} \begin{pmatrix} \psi_{i+k}^{(k-1)} \\ \psi_i^{(-k+1)} \end{pmatrix}. \end{aligned}$$

The kth rows of L^{-1} and U^{-1} are respectively given by $\psi_k^{(k-1)}$ and $\psi_k^{(-n+k)}$, and the kth diagonal elements of $\tilde{U}$ and $\tilde{L}$ by $d_k^{(k-1)}$ and $d_k^{(-n+k)}$.

Because it computes the same quantities $\alpha_{i,i+k}$ and $\beta_{i+k,i}$, hence the same matrix Q, as the generalised Bareiss algorithm, the generalised Levinson algorithm does not break down as long as all contiguous principal submatrices are non-singular. Moreover, forward elimination and backsubstitution can be done in the same manner as for the generalised Bareiss algorithm.

For a symmetric positive-definite matrix A, the generalised Levinson algorithm determines L^{-1}, U^{-1}, D_{L} and D_{U}. Alternatively, the Hyperbolic Cholesky algorithm can be used to derive a symmetric version of the generalised Levinson algorithm that computes the Cholesky factorisations $A^{-1} = \mathcal{U}^{-T}\mathcal{U}^{-1} = \mathcal{L}^{-T}\mathcal{L}^{-1}$ of the inverse.

5.2. Specialisation to Toeplitz Matrices

If the matrix T at hand is a Toeplitz matrix then the generalised Levinson algorithm reduces to the Levinson algorithm [12]. The Levinson algorithm is derived by applying to the Bareiss algorithm the relations

$$\psi_{i+k}^{(k)} = a_{i+k}^{(k)} A^{-1}, \qquad \psi_i^{(-k)} = a_i^{(-k)} A^{-1}, \qquad 1 \le i \le n-k.$$

Hence, only the vectors $\psi_{k+1}^{(k-1)}$, $\psi_n^{(k-1)}$, $\psi_1^{(-k+1)}$ and $\psi_{n-k}^{(-k+1)}$ are needed; see Section 3.4. Since the generalised Levinson algorithm computes the same multipliers $\alpha_{i,i+k}$ and $\beta_{i+k,i}$ as the generalised Bareiss algorithm, the Toeplitz case simplifies to

$$\alpha_k \equiv \alpha_{i,i+k} = \psi_{k+1}^{(k-1)} Te_1 / \psi_1^{(-k+1)} Te_1, \qquad \beta_k \equiv \beta_{i+k,i} = \psi_1^{(-k+1)} Te_{k+1} / \psi_{k+1}^{(k-1)} Te_{k+1}.$$

In contrast to the Bareiss algorithm, which requires the three different elements $a_{k+1,1}^{(k-1)}$, $a_{11}^{(-k+1)}$ and $a_{1,k+1}^{(-k+1)}$, only one set of quantities, $\psi_{k+1}^{(k-1)}$ and $\psi_1^{(-k+1)}$ for instance, suffices to compute *both* multipliers for the Levinson algorithm. Hence, about half of the arithmetic operations in the Levinson algorithm can be saved when only one of the factorisations is desired (that is, the computation of the vectors with subscript n in the algorithm below can be omitted). For simplicity, we write $\phi_1^{(k-1)} \equiv \psi_{k+1}^{(k-1)}$ and $\phi_n^{(-k+1)} = \psi_{n-k}^{(-k+1)}$, and we also exploit the fact that only elements $i, \ldots, i+k$ of $\psi_{i+k}^{(k)}$ and $\psi_i^{(-k)}$ may be non-zero.

Factorisation in the Levinson Algorithm

```
1 ≤ i ≤ n,    $\psi_1^{(0)} = \phi_1^{(0)} = \psi_n^{(0)} = \phi_n^{(0)} = 1, \quad d_1 = t_0$
1 ≤ k ≤ n − 1,
    $\alpha_k = \phi_1^{(k-1)} (t_{-1} \ \ldots \ t_{-k})^T / d_k, \qquad \beta_k = \psi_1^{(-k+1)} (t_1 \ \ldots \ t_k)^T / d_k$
    $d_{k+1} = (1 - \alpha_k \beta_k) d_k$
    $\begin{pmatrix} \phi_1^{(k)} \\ \psi_1^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} 0 & \phi_1^{(k-1)} \\ \psi_1^{(-k+1)} & 0 \end{pmatrix}$
    $\begin{pmatrix} \psi_n^{(k)} \\ \phi_n^{(-k)} \end{pmatrix} = \begin{pmatrix} 1 & -\alpha_k \\ -\beta_k & 1 \end{pmatrix} \begin{pmatrix} 0 & \psi_n^{(k-1)} \\ \phi_n^{(-k+1)} & 0 \end{pmatrix}.$
```

The kth rows of L^{-1} and U^{-1} are respectively given by

$$\begin{pmatrix} \phi_1^{(k-1)} & 0_{n-k} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0_{k-1} & \phi_n^{(-n+k)} \end{pmatrix},$$

and the kth diagonal elements of $\tilde{U}$ and $\tilde{L}$ by d_k and d_{n-k+1}.

If the matrix T is symmetric positive-definite then $\psi_1^{(-k)} = \phi_1^{(k)} J$ and $\psi_n^{(k)} = \phi_n^{(-k)} J$, and because of L=JUJ also $\phi_n^{(-k)} = \phi_1^{(k)} J$. Hence the algorithm simplifies as follows.

Factorisation in the Levinson Algorithm for Symmetric Positive-Definite Matrices

```
1 ≤ i ≤ n,    $\phi_1^{(0)} = 1, \quad d_1 = t_0$
1 ≤ k ≤ n − 1,
    $\rho_k = \phi_1^{(k-1)} (t_1 \ \ldots \ t_k)^T / d_k, \qquad d_{k+1} = (1 - \rho_k^2) d_k$
    $\phi_1^{(k)} = \begin{pmatrix} 1 & -\rho_k \end{pmatrix} \begin{pmatrix} 0 & \phi_1^{(k-1)} \\ \phi_1^{(k-1)} J & 0 \end{pmatrix}.$
```

6. A Parallel Algorithm for Persymmetric Systems

During the usual process of solving a linear system $Ax = b$ as described in Section 2.1, the factorisation and forward elimination part can be performed simultaneously. The backsubstitution part, however, cannot be overlapped with the other two because the first step in the solution of the upper triangular system requires the last element obtained from the solution of the lower triangular system in the forward elimination.

We will now show how one can combine the Bareiss and Levinson algorithms in order to perform all three steps of solving a linear system $Ax = b$ in parallel, where A is a persymmetric matrix. It appears that this algorithm when applied to Toeplitz matrices is identical to one given in [8], but we give a much simpler description here.

Suppose that there are enough processors, i.e. $O(n^2)$, available; and that a division, or a multiplication followed by an addition constitutes one arithmetic operation. The computation

$$Q\begin{pmatrix} T & I & b \\ T & I & b \end{pmatrix} = \begin{pmatrix} \tilde{U} & L^{-1} & c \\ \tilde{L} & U^{-1} & d \end{pmatrix}, \qquad \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} L^{-1}b \\ U^{-1}b \end{pmatrix},$$

where the multipliers α_k and β_k are computed as in the Bareiss algorithm, performs the factorisation and forward elimination. It requires $2(n-1)$ parallel operations because the computation of the multipliers cannot be overlapped with their application.

Because $T = UD_{\mathrm{U}}L_{\mathrm{U}} = LD_{\mathrm{L}}U_{\mathrm{L}}$ and because of persymmetry we actually have

$$\begin{pmatrix} \tilde{U} & L^{-1} & c \\ \tilde{L} & U^{-1} & d \end{pmatrix} = \begin{pmatrix} D_{\mathrm{L}}U_{\mathrm{L}} & L^{-1} & c \\ D_{\mathrm{U}}L_{\mathrm{U}} & U^{-1} & d \end{pmatrix} = \begin{pmatrix} D_{\mathrm{L}}U_{\mathrm{L}} & L^{-1} & c \\ JD_{\mathrm{L}}L^TJ & JU_{\mathrm{L}}^{-T}J & d \end{pmatrix},$$

and $x = \tilde{U}^{-1}c = U_{\mathrm{L}}^{-1}D_{\mathrm{L}}^{-1}c$. Because the Levinson algorithm delivers the elements of D_{L}, the divisions $y = D_{\mathrm{L}}^{-1}c$ can be performed concurrently with the forward elimination (possibly with a lag of $O(1)$ steps). Note that the elements of the vectors c and y are computed in the order $1, \ldots, n$, and the rows of $JU_{\mathrm{L}}^{-T}J$ in the order $n, \ldots, 1$. Consequently, the rows of $U_{\mathrm{L}}^{-T}J = (JU_{\mathrm{L}}^{-1})^T$ are available in the order $1, \ldots, n$, so the columns of JU_{L}^{-1} are available in the order $1, \ldots, n$ (the premultiplying matrix J just permutes the rows and has no effect on the columns). Thus, the ith column of U_{L}^{-1} is available by the time the ith element of y has been computed so that the linear combination $x = U_{\mathrm{L}}^{-1}y$ of the columns of U_{L}^{-1} can be performed concurrently with the factorisation and forward elimination in $2n + O(1)$ parallel operations.

The algorithm for Toeplitz matrices can be implemented on $O(n)$ processors [8] in $2n$ parallel operations, but seems to require a considerable amount of broadcasting. If a truly systolic implementation without broadcasting is desired then we believe that this algorithm has the same time complexity as the one in [4], which uses the matrix Q to perform backsubstitution, see Section 3.3. At last note that the asymptotic time complexity is the same for persymmetric and for Toeplitz systems.

Acknowledgements

I would like to thank Jean-Marc Delosme, Liz Jessup and Jin-Hong Ma for their careful reading of the paper.

References

[1] Ahmed, H.M., Delosme, J.-M. and Morf, M., *Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing,* IEEE Computer, 15 (1982), pp. 65–82.

[2] Bareiss, E.H., *Numerical Solution of Linear Equations with Toeplitz and Vector Toeplitz Matrices,* Numer. Math., 13 (1969), pp. 404–24.

[3] Bunch, J.R., *Stability of Methods for Solving Toeplitz Systems of Equations,* SIAM J. Sci. Stat. Comput., 6 (1985), pp. 349–64.

[4] Delosme, J.-M. and Ipsen, I.C.F., *Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations,* Linear Algebra and its Applications, 77 (1986), pp. 75–111.

[5] ———, *From Bareiss' Algorithm to the Stable Computation of Partial Correlations,* Journal of Computational and Applied Mathematics, 27 (1989), pp. 53–91.

[6] Delsarte, P., Genin, Y. and Kamp, Y., *A Method of Matrix Inverse Triangular Decomposition, Based on Contiguous Principal Submatrices,* Linear Algebra and its Applications, 31 (1980), pp. 199–212.

[7] ———, Generalized Schur Positivity Test and Levinson Recursion, *Proc. Europ. Conf. Circuit Theory and Design,* 1983, pp. 321–3.

[8] Gohberg, I., Kailath, T., Koltracht, I. and Lancaster, P., *Linear Complexity Parallel Algorithms for Linear Systems of Equations with Recursive Structure,* Linear Algebra and its Applications, 88/89 (1987), pp. 271–315.

[9] Golub, G.H. and van Loan, C.F., *Matrix Computations,* The Johns Hopkins Press, 1983.

[10] Ipsen, I.C.F., Systolic Algorithms for the Parallel Solution of Dense Symmetric Positive-Definite Toeplitz Systems, *Numerical Algorithms for Modern Parallel Computer Architectures,* Springer Verlag, 1988, pp. 85–108.

[11] Lawson, C.L. and Hanson, R.J., *Solving Least Squares Problems,* Prentice Hall, 1974.

[12] Levinson, N., *The Wiener RMS (Root-Mean-Square) Error Criterion in Filter Design and Prediction,* J. Math. Phys., 25 (1947), pp. 261–78.

[13] Schur, I., *Ueber Potenzreihen die im Innern des Einheitskreises Beschraenkt Sind,* J. Reine Angewandte Mathematik, 147 (1917), pp. 205–32.